Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Курсова робота

освітній ступінь – бакалавр

на тему: «**Frontend розробка на основі React**»

Виконав: студент 3-го року навчання,

Спеціальності

121 «Інженерія Програмного Забезпечення»

Студента Богути Станіслава

Керівник Бабич Т.А.

магістр комп’ютерних наук, асистент

«30» травня 2022 р.

Київ – 2022

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Освітній ступінь бакалавр

Спеціальність 121 «Інженерія Програмного Забезпечення»

Освітня програма бакалавр

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

Гороховський С. С.

“12” жовтня 2021 року

ЗАВДАННЯ
ДЛЯ КУРСОВОЇ РОБОТИ СТУДЕНТУ

Богуті Станіславу

1. Тема роботи **«Frontend розробка на основі React»**, керівник роботи Бабич Трохим Анатолійович, магістр комп’ютерних наук, асистент

2. Строк подання студентом роботи 17 травня 2022

3. План роботи

Анотація

Вступ

Розділ 1. Дослідження та аналіз предметної області

1.1. Структура frontend застосунку

1.2. SPA та MPA

1.3. Client-side та server-side рендеринг

1.4. Комунікація у розподілених системах

1.5. Засоби часткового провантаження сторінок

ГРАФІК ПІДГОТОВКИ КУРСОВОЇ РОБОТИ ДО ЗАХИСТУ

№ з/п	ПЕРЕЛІК РОБІТ	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітки
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника Узгодження календарного графіка підготовки кваліфікаційної роботи. Ознайомлення студента з критеріями оцінювання кваліфікаційної роботи (п. 8.5).	12 жовтня 2021			
2.	Вивчення джерел літератури, матеріалів архівів, періодичних видань, збір та узагальнення фактів, даних	12 жовтеня 2021 – 16 листопада 2021			
3.	Складання плану каліф. роботи та узгодження з науковим керівником	17 листопада 2021			
4.	Написання розділів роботи	2 листопада 2021 – 01 березня 2022			
5.	Проміжний контроль виконання роботи	01 лютого 2022			
6.	Написання кваліфікаційної роботи в цілому, ознайомлення з її першим варіантом наукового керівника	11 січня 2022 – 29 березня 2022			
	Розділ 1 (постановка проблеми, теоретичні основи, огляд літературних джерел)	25 січня 2022			
	Розділ 2 (аналітично-дослідницька частина)	01 березня 2022			
	Розділ 3 (проектно-рекомендаційна частина)	29 березня 2022			
7.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	01 квітня 2022 – 06 травня 2022			
8.	Подання кваліфікаційної роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності,	17 травня 2022			
9.	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	згідно з розкладом роботи ЕК			

Графік узгоджено 10 жовтня 2021 р.

Науковий керівник Бабич Трохим Анатолійович

Виконавець курсової роботи Богута Станіслав

ЗМІСТ

Анотація

У даній роботі розглядаються принципи розробки та види frontend застосунків. Детально розглядається та пояснюється вибір React для розробки frontend застосунків. На прикладі розробленого застосунку розглядаються проблеми написання застосунку та шляхи їх вирішення за допомогою React та додаткових бібліотек, які працюють разом із React. Детально розглядається використання компонентів для створення ефективного та підтримуємого застосунку, який надає весь необхідний функціонал кінцевому користувачу.

Вступ

Поточними тенденціями всього світу зараз є діджиталізація. Все більше компаній, закладів, бізнесів хочуть мати свій web-застосунок, який буде їх представляти всім користувачам інтернету. Паралельно з необхідністю створювати сайти постає необхідність швидкого створення сайтів. При цьому застосунки не повинні створювати погане враження на користувача, повинні бути ефективними та надійними. Тобто ми маємо набір проблем, які треба швидко та ефективно вирішувати задля просування в інтернеті та заохочення користувачів. Всі ці проблеми напряму визначають прибутковість бізнесу, яка формує його розвиток. Саме тому вивчення технологій, які дозволяють швидко та ефективно розробити web-застосунок є необхідним на даний момент.

Метою цієї роботи є дослідити різні види та підходи до frontend розробки. Провести порівняльний аналіз різних підходів, а також розробити свій frontend застосунок, який буде вирішувати нагальні проблеми користувача.

Було виокремлено такі завдання дослідження та розробки:

- Визначити з чого складається будь-який frontend застосунок
- Дослідити типи frontend застосунків, їх переваги та недоліки
- Дослідити підходи до використання ресурсів систем під час використання fronend застосунку
- Дослідити процеси комунікації у системах, частиною яких є frontend застосунок
- Дослідити та описати бібліотеку React, основу ідею та підхід у використанні при розробці
- Визначити та описати предметну область для розроблюємого застосунку

- Розробити frontend застосунок базуючись на досліджених принципах та підходах до розробки, а також описати проблеми, які можуть виникати під час розробки

Розділ 1. Дослідження та аналіз предметної області

1.1. Структура frontend застосунку

Frontend – частина web-додатків із якою може взаємодіяти кінцевий користувач напряму. Основними задачеми frontend частини є відображення функціонального інтерфейсу призначеного для обробки запитів кінцевого користувача.

Web-додаток – додаток який складається із двох частин: клієнту та серверу. Клієнтом переважно є браузер, а сервером є web-сервер. Відповідно логіка всього додатку розподілена між цими двома частинами, а комунікація між ними відбувається через мережу.

Розробка frontend застосунків відбувається за допомогою трьох основних компонентів: HTML, CSS, JavaScript.

HTML (hypertext markup language) – мова розмітки документів, які відображаютья у браузері.

CSS (cascading style sheets) – мова стилізування сторінок, яка використовується у комбінації із HTML для наданню сторінок необхідного вигляду.

JavaScript – мова програмування, яка використовується у web-розробці. За допомогою неї реалізується клієнтська логіка (анімації, валідація, навігація тощо), а також може бути реалізована серверна логіка.

Всі ці три структурні компоненти розуміє будь-який браузер. Таким чином немає необхідності писати різні застосунки для користувачів із різними браузерами, що спрощує процес розробки. Але слід зауважити, що технології оновлюються і деякі нові можливості HTML та CSS можуть підтримуватися не у всіх версіях всіх браузерів. Тому при використанні певних можливостей слід перевіряти їх наявність у необхідних браузерах.

1.2. SPA та MPA

MPA (multi-page application) – концепція розробки web-застосунків, у якій кожна сторінка сайту є окремим файлом на сервері та завантажується при необхідності. Тобто якщо на сайті є декілька сторінок, то при переході користувача з однієї сторінки на іншу відбувається запит від клієнта до сервера метою якого є отримання всіє нової сторінки.

З переваг MPA можна виділити гарну можливість для SEO оптимізації, легкість скейлінгу та інтеграцію із застосунками для аналітики. У свою чергу такі застосунки можуть мати проблеми із продуктивністю, підтримкою та оновленням через велику кількість сторінок.

SPA (single-page application) – концепція розробки web-застосунків, у якій користувач отримує з серверу лише одну сторінку і потім всі його взаємодії з нею реалізуються за допомогою клієнтського JavaScript. У такій концепції немає необхідності щоразу підвантажувати новий html документ, а достатньо провантажити всі файли 1 раз, та потім взаємодіяти із ними.

З переваг SPA можна виділити гарну продуктивність, гарний досвід користувача, кешування даних та швидкість розробки. Серед недоліків виділяються проблеми із SEO, час першого провантаження та необхідність обов’язкової підтримки JavaScript зі сторони користувача.

1.3. Client-side та server-side рендеринг

Server-side рендеринг – тип провантаження сторінки у якому всі необхідні операції зі створення HTML сторінки відбувається на стороні серверу. Тобто користувач лише посилає запит певної сторінки сайту на сервер, а той в свою чергу повністю будує HTML документ та повертає його користувачеві.

З переваг server-side рендерингу можна виділити грану швидкість провантаження сторінки для одного юзера та SEO оптимізацію.

Client-side рендеринг – тип провантаження сторінки при якому переважна частина операцій щодо провантаження лягає на сторону користувача. При такому типі провантаження браузер отримує від серверу лише частину HTML документа та набір JavaScript файлів, які і відповідають за провантаження сторінки.

Серед переваг client-side ренедрингу можна виділити швидкість готовності сторінки, менше навантаження на сервер та швидкість розробки. Недоліками же є важкість SEO оптимізації та важке перше провантаження сторінки.

1.4. Комунікація у розподілених системах

Frontend застосунок використовується лише для представлення та отримання інформації від кінцевого користувача. Для обробки інформації та її збереження на сервері використовується певний backend застосунок. Для комунікації між цими двома застосунками використовуються RESTful API.

REST – це стиль архітектури програмного забезпечення для розподілених систем. У такій архітектурі комунікація між різними застосунками відбувається за допомогою HTTP запитів. Такий тип комунікації характеризується відсутністю необхідності розуміння стану користувача зі сторони сервера. Тобто запит до сервера містить всю необхідну інформацію для його коректної обробки.

HTTP – це протокол передачі інформації який використовується у мережі. Кожен запит за таким протоколом складається з типу запиту та url. Основні типи запитів це: GET, POST, PUT, DELETE. Відповідно кожен тип запиту

має своє логічне призначення, наприклад, GET використовуються для отримання певних даних, а POST – для додавання. Кожен запит може мати набір параметрів, які необхідні для коректної обробки запиту зі сторони сервера. Параметри можуть передаватися як параметри запиту, тобто у рядочку HTTP запиту до сервера, або у тілі запиту (якщо це метод POST).

RESTful API – інтерфейс на стороні серверу, який дозволяє оброблювати REST запити.

Завдяки REST архітектурі frontend та backend додатки можуть швидко обмінюватися інформацією, а також вони є масштабованими. Масштабованість є важливою для web-додатків, оскільки існує необхідність коректного відпрацювання навіть при великому навантаженні на додаток. Збільшення кількості екземплярів додатку дозволяє розподілити навантаження, а REST комунікація забезпечить коректний обмін інформацію навіть для декількох екземплярів застосунку.

1.5. Засоби часткового провантаження сторінок

React – це відкрита бібліотека JavaScript, яка створення для вирішення проблем часткового провантаження сторінок при розробці. Основною ідеєю цієї бібліотеки є декларативність та гнучкість розробки, яка забезпечується за допомогою компонентів.

Переважно на React реалізують концепцію SPA та client-side рендеринг. Відповідно до цього, сторінки створенні за допомогою цієї бібліотека провантажуються за допомогою JavaScript у браузері користувача, та самі застосунки вимагають широкої взаємодії із DOM сайту. Також, за допомогою фреймворку Next.js можна на React реалізувати server-side рендеринг. При такому використанні, всі необхідні html сторінки будуть

генеруватися за допомогою Next.js на сервері із мінімальним набором додаткових JavaScript та CSS файлів, а потім відправлятися юзеру.

DOM (document object model) – програмний інтерфейс для web-документу. Цей інтерфейс перетворює сторінку на певний об’єкт з яким може взаємодіяти через програмний код, наприклад за допомогою JavaScript.

React компонент – структурний об’єкт JavaScript коду, який описує певну частину сторінки. У компоненті прописана вся його логіка та поведінка.

JSX – розширення JavaScript, яке дозволяє прописувати HTML компоненти у звичайному JavaScript коді. JSX використовується у React для опису його компонентів.

Компоненти у React бувають двох типів:

- Підкласові компоненти
  Компонент у React можна описати за допомогою класів. Для цього потрібно оголосити клас, який наслідує клас React.Component. В класу можна описати всю необхідну логіку компонтента. Метод render() – головний метод компоненту, який повертає полегшену версію цього ж самого компонента. Важливо розуміти, що такий компонент має власний стан.
- Функціональні компоненти
  Компоненти у React можна також описати за допомогою функцій. Такі компоненту на відміну від підкласових не мають власного стану. Компонент лише отримує необхідні дані та відображають їх в описаний спосіб. Також у функціональних компонентів відсутній метод render(), замість нього опис компонента повертається через return().

Відповідно функціональні та класові компоненти мають різне призначення, та різний синтаксис для опису.

Використання React компонентів відбувається у декларативний спосіб. Необхідно прописати тег, який називається як відповідний React компонент. Також у тег можна передавати відповідні параметри за їх назвою.

У React застосунку одні компоненти використовують інші компоненти, таким чином все зводиться до того, що точкою входу до такого застосунку є один файл index.js у якому описане використання всіх компонентів.

Однією з переваг React є використання стану для компонентів. Для зручної роботи із станами використовують бібліотеку React Redux. Ця бібліотека впроваджує сховище, яке використовується для зручного зберігання та зміни станів об’єктів у застосунку.

React широко поширений завдяки своїм перевагам:

- Віртуальний DOM

  Оскільки React працює на основі динамічного перебудування DOM браузера, то для оптимізації цього процесу він використовує технологію віртуального DOM. Віртуальний DOM дозволяє React зберігати у пам’яті та швидко міняти поточний стан сторінки, а потім синхронізувати його із реальним DOM браузера за допомогою API браузера.
- Перевикористання компонентів
- SEO-friendly

  React має гарну інтеграцію із пошуковою системою Google, що дозволяє добре SEO оптимізовувати сайти.
- Широка підтримка

Для зручної роботи із React було створено низку інструментів та бібліотек, які допомагають при розробці. Прикладом такого застосунку є розширення React Developer tools у Google Chrome, яке допомагає зручно працювати із React.

1.6. Висновки

Отже, загальна структура будь-якого frontend застосунку складається із трьох основних компонентів: html, js, css. Відповідно кожний компонент відповідає за свою частину сторінки. Існують різни види frontend застосунків такі як SPA та MPA, кожна з яких має свої переваги та недоліки. Також, при розробці frontend застосунку треба пам’ятати про різні режими рендеру сторінки, а саме client-side та server-side рендеринг.

Для швидкої та якісної розробки слід використовувати новітні технології, наприклад React. Завдяки готовим технологічним рішенням розробник може не зациклюватися на проблемах, рішення для яких вже існує, а концентруватися на реалізації необхідної йому застосунку.

Розділ 2. Проектування та розробка системи

2.1. Постановка задачі та опис предметної області

За предметну область було обрано бізнес процеси при роботі закладів. В процесі роботи кожного закладу є такі елементи як створення замовлення, перегляд історії замовлень, різна статистика, моніторинг існуючих продуктів на складах тощо. Тому для зручності та оптимізації відповідних процесів було вирішено створити інтерфейс адміністратора.

У ході аналізу предметної області було виділено наступні компоненти та типи взаємодії із ними:

- Склад – сутність, яка описує певний склад закладу. Користувач повинен мати можливість редагувати, створювати, видаляти склад, а також переглядати історію операцій пов’язаних із цим складом.
- Категорії продуктів – сутність, яка описує різні категорії продуктів які використовуються у закладі. Користувач повинен вміти створювати, редагувати, видаляти категорію, а також переглядати список продуктів, які відносяться до певної категорії.
- Продукт – сутність, яка описує певний продукт, який використовує заклад. Користувач повинен вміти створювати, редагувати, видаляти продукти, а також переглядати список операцій для кожного продукту.
- Технічна карта – сутність, яка описує певну позицію меню закладу. У цій сутності зберігається інформація про продукти, які використовуються для створення одної одиниці продукту, та інші поля необхідні для опису позиції меню. Користувач повинен мати можливість створювати, редагувати, видаляти певну позицію, переглядати список відповідних продуктів та історію взаємодії із позицією.

- Категорії технічних карт – сутність, яка описує категорії до яких можна віднести певну технічну карту. Користувач повинен мати можливість створювати, редагувати, видаляти певну категорію, а також переглядати технічні карти відповідної категорії.
- Провайдер – сутність, яка описує організацію (людину), яка постачає необхідні продукти на склад. Користувач повинен мати можливість створювати, редагувати, видаляти провайдерів, а також переглядати історію операцій певного провайдеру.
- Менеджмент операцій на складах – сутність, яка описує дії пов’язані із операціями на складах. Користувач повинен вміти створювати, редагувати, видаляти дії, а також переглядати список продуктів відповідної операції.
- Менеджмент клієнтів – сутність, яка описує клієнта закладу. Користувач повинен мати можливість створювати, редагувати, видаляти клієнта, а також переглядати історію замовлення певного клієнта.
- Дискон – сутність, яка описує певну знижку (акцію) закладу. Користувач повинен вміти створювати, редагувати, видаляти певний дисконт, а також переглядати список операцій та список технічних карт за дисконтом.
- Замовлення – сутність, яка описує замовлення у закладі. Користувач повинен мати можливість переглядати, створювати, редагувати, підтверджувати замовлення.

Отже, метою буде створити інтерфейс користувача, який буде надавати необхідні можливості для користувачів даної предметної області.

2.2. Вибір технології

Для реалізації поставленої задачі було вирішено обрати бібліотеку React, та сам застосунок реалізовувати як SPA із client-side рендерингом. Оскільки даний застосунок передбачений для використання у закладах, то немає необхідності SEO оптимізації, важкість якої є відповідним недоліком обраного набору технологій. Застосунки які використовують рендеринг зі сторони клієнта працюють швидко, що є важливим для бізнес процесів закладу при великому потоці клієнтів. Також важливим фактором використання React стала швидкість розробки застосунку.

Для створення та використання React застосунку використовується Node Package Manager (npm). NPM – це менеджер бібліотек для JavaScript. Дозволяє у зручний спосіб встановлювати та використовувати необхідні бібліотеки.

2.3. Вибір бібліотеки компонентів

React надає лише підхід для створення frontend застосунків. Основою цього підходу, як зазначалося, є ідея компонентів сторінки. Відповідно, для швидкої розробки, та якісного досвіду використання застосунку зі сторони користувача, важливим фактором є створення та використання компонентів інтерфейсу. Оскільки React існує відносно довго, то вже існують набори бібліотек, які надають певні функціональні компоненти.

Для реалізації застосунку було обрано бібліотеку Ant Design. Бібліотека надає різні компоненти для введення та представлення інформації на сторінці, а також пропонує свої зручні компоненти для навігації та розмітки сторінки. Важливим фактором використання цієї бібліотеки є частота оновлення та підтримка різних браузерів. Наразі Ant Design підтримує найпопулярніші браузери такі як Chrome, Firefox, Safari, Opera та Edge.

Додавання Ant Design у React проект відбувається за допомогою відповідної npm команди. А імпорт та використання компонентів бібліотеки із бібліотеки відбувається у коді. (див. Додаток А)

2.4. Декларування та використання станів

При написанні застосунку важливою проблемою постає розуміння системою стану користувача. Тобто для навігації користувача на відповідну сторінку, або виконання необхідної дії, система повинна розуміти де зараз знаходиться користувач та чи дозволено йому робити ту дію, яку він хоче. Для цього для React існує бібліотека Redux, яка і дозволяє централізовано управляти станами користувача на сторінці. Встановлення Redux відбувається через npm.

Кожний стан користувача описується у окремому слайсі, де зазначається назва, поля, початкові значення полів та відповідні дії, які можна робити із цим станом. Слайс – об’єкт який описує певний стан. Дії визначені для певного стану називаються редюсерами, та описуються за допомогою функцій. Лише через відповідні редюсери можна змінити певний стан користувача. Тобто, якщо правильно описати всі дії можливі із певним станом, то при подальшій зміні стану достатньо звертатися до відповідного редюсера. Такий підхід дозволяє при розробці випадково не змінити стан користувача на такий, який не може бути під час використання системи.

Після того як описані всі необхідні слайси, всі редюсери об’єднуються у один об’єкт і передаються у сховище користувача. Сховище користувача – об’єкт у якому зберігається вся інформація про стани та відповідні дії із ними для даного користувача. Таке сховище є унікальним для кожного користувача застосунку.

При розробці застосунку, було описано стани користувача відповідно до кожної сторінки, на якій користувач може бути. Це дозволяє зберігати стан користувача на сторінці навіть після того, як він перейшов на іншу сторінку та повернувся назад. Тобто, якщо користувач займався створення замовлення, а потім пішов подивитися історію клієнта, то після повернення на сторінку замовлень користувач все що буде у стані створення замовлення. Для підключення Redux станів до свого застосунку необхідно весь всі компоненти розмістити в середині Provider компонента Redux бібліотеки. (див. Додаток Б)

2.5. Робота із локальним сховищем станів

Проблемою використання Redux сховище в чистому виді є його тимчасовість. Стани зберігаються у тимчасовому сховищі поточної сесії сторінки, через що після перезавантаження стан перестворюється і користувач починає всю взаємодію із застосунком з самого початку.

Для вирішення цієї проблеми було використано бібліотеку Redux persist, вона дозволяє зберігати стан користувача у постійне сховище, які підвантажується у застосунок за використання. Таким чином користувач може залишатися у стані який був до перезавантаження або перевикористання застосунку.

Для використання постійного сховище до застосунку, було створено відповідну конфігурацію сховища та підключено за допомогою PersistGate компонента. (див. Додаток В)

2.6. Навігація в середині застосунку

У кінцевого користувача обраної предметної області є необхідність взаємодіяти із різними елементами бізнес логіки. Для цього кожен такий елемент був винесений на окрему сторінку для кращого досвіду

використання. У той же момент з’явилася необхідність створення логіки для переходу між сторінками. Слід зазначити, що в застосунку використовується концепція SPA та client-side рендерингу, через яку немає явного поділу на сторінки, весь застосунок є однією сторінкою на якій перебудовується DOM структура, тому для користувача це виглядає як перехід між сторінками.

Для реалізації переходу між сторінками було використано бібліотеку react-routed-dom. Вона надає Route компонент, який дозволяє динамічно змінювати url сторінки та промальовувати необхідні компоненти в залежності від сторінки. Таким чином застосунок має явне розділяння між сторінками, кожна з яких має свій url. (див. Додаток Г)

2.7. Використання компоненти вищого рівня

Компоненти вищого рівня – підхід до створення компонентів у React основною метою якого є провантаження необхідного компонента за певних умов.

Необхідність використання компонентів вищого рівня з’явилася при створенні авторизації у застосунку. Ми не хочемо щоб авторизований користувач мав доступ до певного функціоналу, тому нам щоразу необхідно перевіряти чи авторизований користувач. Для такої перевірки був використаний компонент вищого рівня, який приймає інший компонент необхідний для відображення за умов авторизації. Таким чином всі компоненти можна обгорнути в компонент вищого рівня та обмети доступ до певних категорій кінцевих користувачів.

Необхідність використання компонентів вищого рівня також виникла під час реалізації функціоналу створення, редагування, видалення для відповідних сутностей. Ідея реалізації полягає в тому, що ми не хочемо щоб

кінцевий користувач сприймав сторінку редагування чи створення як абсолютно іншу сторінку. Тобто користувач повинен думати, що функціонал створення чи редагування це частина сторінки відповідної сутності. При цьому в коді ми хочемо відрізняти компоненти для редагування та створення, щоб потім було легше їх змінювати. Тому саме для відображення відповідного функціоналу як певної частини сторінки були використані компоненти вищого рівню. У даній реалізації компоненти вищого рівню не приймають на вхід якийсь компонент із зовні, а відображають необхідний компонент на основі поточного стану користувача. Саме такі компоненти вищого рівню використовуються у компоненті Route, який відповідає за розділення сторінок для кінцевого користувача. (див. Додаток Ґ)

2.8. Взаємодія frontend застосунку із backend застосунком

Frontend застосунку необхідно обмінюватися інформацією backend застосунком, оскільки саме backend відповідає за оновлення інформацій у базі даних, а frontend відповідає за введення та відображення інформації. Для взаємодії цих застосунків було обрано REST комунікацію. Відповідно backend має RESTful API через яке frontend може посилати та отримувати інформацію.

У frontend застосунку є необхідність отримувати інформацію при завантаженні певних сторінок, на приклад сторінки із замовленнями. Для цього у React компонентах використовується useEffect метод. useEffect – стандартний метод функціонального компоненту React. Він виконується після того, як об’єкт був промальований на сторінці. Саме через useEffect можна відправити запит на сервер, почекати необхідну інформацію, а потім перемалювати об’єкт в залежності від відповіді серверу. Таким чином ми

можемо отримати інформацію від сервера, наприклад, про замовлення та відобразити їх кінцевому користувачу.

Для відправлення REST запиту у JS використовується метод fetch. Fetch – це метод об’єкта window (вікна браузера користувача), який робить AJAX запити. AJAX запити – це тип запитів, які робиться на сторінці клієнта у фоновому режимі, що дозволяє отримувати та промальовувати дані сервера без перезавантаження сторінки.

Даний підхід дозволяє не турбуватися про постійну присутність необхідних даних на сторінці після їх оновленні на сервері. Застосунок сам буде підвантажувати та відображати нові дані, що створить якісний досвід використання застосунку. (див. Додаток Д)

Висновки

Отже, в написання роботи було всесторонньо досліджену побудову frontend застосунку. Розглянуто різні види frontend застосунків, такі як Single Page Application та Multi Page Application. Також розглянуто та порівняно переваги та недоліки відповідних видів застосунків. Розглянуто та порівняно підходи щодо створення сторінок у застосунку, а саме client-side та server-side створювання. Розглянуто проблему часткової зміни сторінки, а також способи вирішення цієї проблеми за допомогою сучасних технологій frontend розробки. Також було розглянуто модель взаємодії розподілених систем, а саме взаємодію frontend та backend застосунків на основі протоколу HTTP та принципів REST. У практичній частині була поставлена задача на реалізацію інтерфейсу для вирішення завдань зазначеної предметної області. Предметна область була сформульована та описана. Було розроблено відповідний React застосунок та розглянуто детальніше проблеми при розробці та шляхи їх вирішення. Було детально описано принципи розробки React застосунку, види компонентів та їх практичне використання для вирішення поставленої задачі. Було розглянуто та використано додаткові інструменти розробки на React такі як Redux та Ant Design. Було описано переваги та недоліки використання React у frontend розробці, а обґрунтовано його вибір для реалізації поставленої задачі.

Список використаних джерел

1. GAVRILĂ V. Modern Single Page Application Architecture: A Case Study [Електронний ресурс] / V. GAVRILĂ, L. BĂJENARU, C. DOBRE. – 2019. – Режим доступу до ресурсу: https://sic.ici.ro/wp-content/uploads/2019/06/Art.-11-Issue-2-SIC-2019.pdf.
2. Deshmukh S. Building a Single Page Application Web Front-end for E-Learning site [Електронний ресурс] / S. Deshmukh, A. Retawade, D. Mane – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/8819703/authors#authors.
3. Analyzing best practices on Web development frameworks: The lift approach [Електронний ресурс] / [M. PilarSalas-Zárate, G. Alor-Hernández, R. Valencia-García та ін.] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S0167642314005735.
4. Application of Restful APIs in IOT: A Revie [Електронний ресурс] / [P. Babbe, K. Nambiar, R. Maurya та ін.] – Режим доступу до ресурсу: https://d1wqtxts1xzle7.cloudfront.net/65937498/33013-with-cover-page-v2.pdf?Expires=1654438634&Signature=GF7T~PA0kcmGR48fhSHgbLqBpwCgxpEBRJNjf-rKUul69YYquZXjnBpOnNLyplp-mL3PBIBuZsNULXQ5Kpl3BsCGq9sI67gwNTjnTe3RVYvzvY3kFvMktnsJNZ70irBme4Xy3tVNhmkpMTquamiZGfvU1qiUMm6-HYiXYYS9DnH-WnjzK49u8p~fTEX~tqOrDi8Lsua65dUv0-dReKK0jQ8nBmJATTmjy2462Hi4uuC9haOd0k93wuGwelZpdNKrywWxzC2WhjGf~5nGQ8TlNMmVqWy21MOFnaWmCzQFIJzOOgTzk-dR5ymSChseVo-JqvBkmntlNj0Uz92C50x6Bw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.
5. Carter B. HTML Architecture, a Novel Development System (HANDS): An Approach for Web Development [Електронний ресурс] / Brian Carter – Режим доступу до ресурсу: https://ieeexplore.ieee.org/document/7113671.
6. Banks A. Learning React: Functional Web Development with React and Redux [Електронний ресурс] / A. Banks, E. Porcello – Режим доступу до ресурсу: https://books.google.com.ua/books?hl=en&lr=&id=ycTADgAAQBAJ&oi=fnd&pg=PP1&dq=react+and+redux&ots=cwWLndA4lP&sig=s0YE6imnVhPjDsq9ObCW0VXFrWY&redir_esc=y#v=onepage&q=react%20and%20redux&f=false.
7. Kudiabor D. T. State management with React-Redux [Електронний ресурс] / Dominic Travis Kudiabor – Режим доступу до ресурсу: https://www.theseus.fi/bitstream/handle/10024/355184/dominic_kudiabor.pdf?sequence=2&isAllowed=y.
8. Purewal S. Learning Web App Development: Build Quickly with Proven JavaScript Techniques [Електронний ресурс] / Semmy Purewal – Режим доступу до ресурсу: https://books.google.com.ua/books?hl=en&lr=&id=JLDZAgAAQBAJ&oi=fnd&pg=PP1&dq=web+app+development&ots=2j6IDHEP8q&sig=UMN-

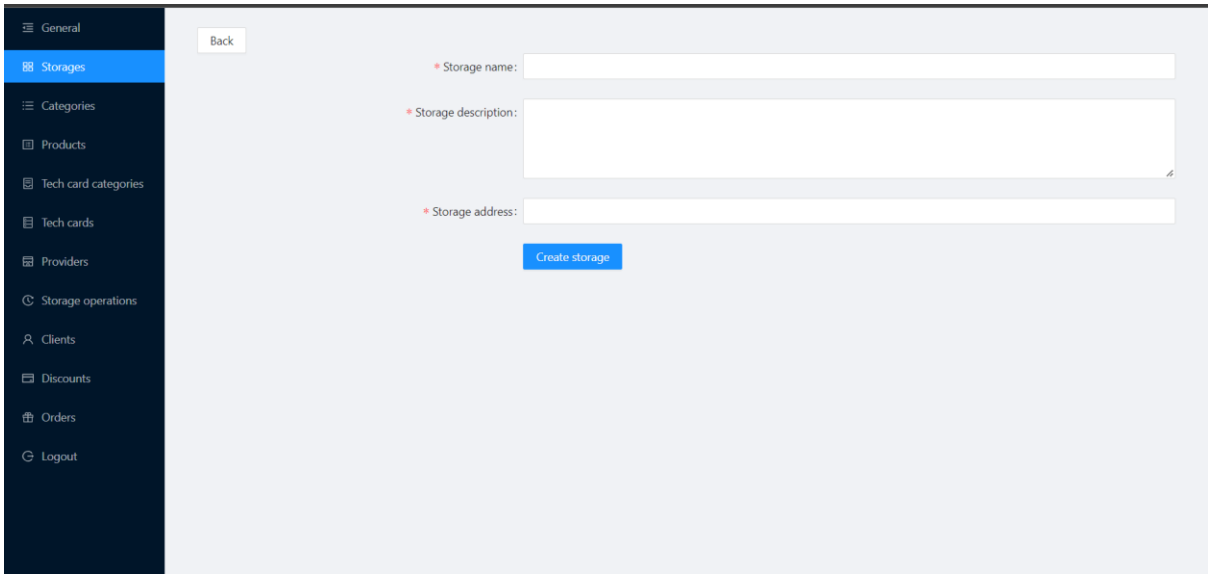
xfqPozMa5Cs3ja_nCmHODWc&redir_esc=y#v=onepage&q=web%20app%20development&f=false.

9. Konshin K. Next.js Quick Start Guide: Server-side rendering done right [Електронний ресурс] / Kirill Konshin – Режим доступу до ресурсу: https://books.google.com.ua/books?hl=en&lr=&id=-rBmDwAAQBAJ&oi=fnd&pg=PP1&dq=javascript+client+side+rendering&ots=xdGLRvuTzf&sig=6kdi5XcAqCcPwrmGbgWn2FEXkxk&redir_esc=y#v=onepage&q=javascript%20client%20side%20rendering&f=false.
10. Coulouris C. Distributed Systems: Concepts and Design [Електронний ресурс] / C. Coulouris, J. Dollimore, T. Kindberg – Режим доступу до ресурсу: https://books.google.com.ua/books?hl=en&lr=&id=d63sQPvBezgC&oi=fnd&pg=PR11&dq=distributed+systems&ots=qysw2mBoWI&sig=RDiD-EsvzNX0xP2IUpVctFJ14ls&redir_esc=y#v=onepage&q=distributed%20systems&f=false.

Додатки

Додаток А

Приклад інтерфейсу на основі Ant Design компонентах



Додаток Б

Використання станів у Redux на прикладі станів сторінки із замовленнями:

- Декларування стану

```
import { createSlice } from '@reduxjs/toolkit'

export const orderSlice = createSlice({
    name: 'order',
    initialState: {
        toEdit: false,
        toInfo: false,
        toCreate: false,
        order: null
    },
    reducers: {
        selectToEdit: (state, order) => {
            state.toEdit = true
            state.order = order.payload

        },
        selectToInfo: (state, order) => {
            state.toInfo = true
            state.order = order.payload

        },
        toCreate: (state) => {
            state.toCreate = true
        },
        remove: (state) => {
            state.toEdit = false
            state.toInfo = false
            state.toCreate = false
            state.order = null
        }
    }
})

export const {selectToEdit,selectToInfo, remove,toCreate,selectToTechCard} =
orderSlice.actions
export default orderSlice.reducer
```

- Використання всіх станів

```
const rootReducer = combineReducers({
    login: loginReducer,
    menu: menuReducer,
    storage: storageReducer,
    productCategory: productCategoryReducer,
    product: productReducer,
    techCardCategory: techCardCategoryReducer,
    techCard: techCardReducer,
    provider: providerReducer,
    discount: discountReducer,
    client: clientReducer,
    order: orderReducer,
    storageOperation: storageOperationReducer
})

  export default rootReducer
```

- Створення сховища

```
import { configureStore } from '@reduxjs/toolkit'
import { persistStore, persistReducer } from 'redux-persist'
import storage from 'redux-persist/lib/storage'
import rootReducer from './reducer'

const persistConfig = {
  key: 'root',
  storage,
}

const persistedReducer = persistReducer(persistConfig, rootReducer)

export default () => {
    let store = configureStore({
        reducer: persistedReducer
    })
    let persistor = persistStore(store)
    return {store, persistor}
}
```

- Підключення станів

```
 <Provider store={store}>
      <PersistGate loading={null} persistor={persistor} >
        <BrowserRouter>
```

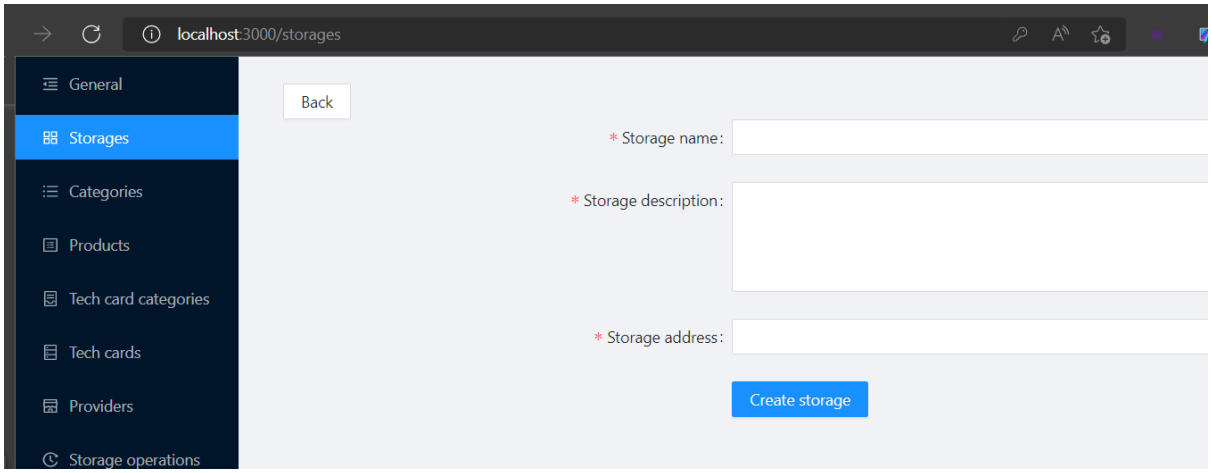
```
            <Layout style={{minHeight: '100vh',}}>
```

Додаток Г

Використання Route компонентів

```
<Routes>
                    <Route exact path="/" element={<MainPage />} />
                    {/* <Route exact path="/login" element={<LoginPage/>} />
*/}
                    <Route exact path="/logout" element={<LogoutPage/>} />
                    <Route path="/storages" element={<StoragePage/>} />
                    <Route path="/categories"
element={<ProductCategoryPage/>} />
                    <Route path="/products" element={<ProductPage/>} />
                    <Route path="/techcard/categories"
element={<TechCardCategoriesPage/>} />
                    <Route path="/techcard" element={<TechCardsPage/>} />
                    <Route path="/providers" element={<ProvidersPage/>} />
                    <Route path="/discounts" element={<DiscountPage/>} />
                    <Route path="/clients" element={<ClientsPage/>} />
                    <Route path="/orders" element={<OrdersPage/>} />
                    <Route path="/storages/operations"
element={<StorageOperationPage/>} />
                </Routes>
```

Приклад відображення адреси сторінки у користувача



Додаток Ґ

Компонент вищого рівня для перевірки авторизації

```
export default function(WrappedComponent){
    class LoginComponent extends React.Component{
        render(){
            if(!this.props.isLogin){
                return(
                    <Navigate to="/" />
                )
            }
            return(
                <WrappedComponent {...this.props}/>
            )
        }
    }

    LoginComponent.propTypes = {
        isLogin: PropTypes.bool.isRequired
    }

    function mapStateToProps(state){
        return{
            isLogin: state.login.value
        }
    }
    return connect(mapStateToProps)(LoginComponent)
}
```

Компонент вищого рівня для відображення функціоналу редагування та створення в межах сторінки:

```
export default function(){
    class LoginComponent extends React.Component{
        render(){
            if(!this.props.isLogin){
                return(
                    <Navigate to="/" />
                )
            }
            if(this.props.toEdit){
                return(
                    <ClientEditComponent {...this.props}/>
                )
```

```
            }
            if(this.props.toCreate){
                return(
                    <OrderCreateComponent {...this.props}/>
                )
            }
            if(this.props.toInfo){
                return(
                    <ClientInfoComponent {...this.props}/>
                )
            }
            return(
                <OrderPageComponent {...this.props}/>
            )
        }
    }

    LoginComponent.propTypes = {
        isLogin: PropTypes.bool.isRequired
    }

    function mapStateToProps(state){
        return{
            isLogin: state.login.value,
            toInfo: state.order.toInfo,
            toEdit: state.order.toEdit,
            toCreate: state.order.toCreate,
            order: state.order
        }
    }
    return connect(mapStateToProps)(LoginComponent)
}
```

Додаток Д

Приклад запиту для отримання списку замовленб

```
useEffect(() => {
        const data = fetch("http://localhost:8090/orders",{
        method: 'GET',
        headers: {
        'Content-Type': 'application/json',
        'Authorization': token
        }
        })
        .then(r => r.json())
        .then(r => {
            setOrders(r);
        })
    })
```

Додаток В

Декларування локального сховища станів

```
const persistConfig = {
  key: 'root',
  storage,
}

const persistedReducer = persistReducer(persistConfig, rootReducer)
```

Використання локального сховища станів

```
    <PersistGate loading={null} persistor={persistor} >
```